ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΑ ΕΡΓΑΛΕΙΑ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ ΓΙΑ ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ

ΜΑΘΗΜΑ ΣΤΟ ΔΠΜΣ : ΕΠΙΣΤΗΜΗ ΔΕΔΟΜΕΝΩΝ & ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

ΜΕΡΟΣ 1: PYTHON

ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ:

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΖΕΡΚΕΛΙΔΗΣ ΔΗΜΗΤΡΙΟΣ

Α.Μ. 003400049

ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΕΙΣΟΔΟ ΣΤΟ ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ

CleanData.py

Στον φάκελο **“ stocks ”** που έχουμε κατεβάσει από το σύνδεσμο που δίνεται στην εκφώνηση της εργασίας, έχουμε όλες τις μετοχές οι οποίες σε αριθμό είναι 7196.

Για αυτόν τον λόγο, επειδή υπάρχει ένα μεγάλο πλήθος μετοχών και λόγω έλλειψης υπολογιστικής δύναμης, δημιουργώ ένα αρχείο το οποίο λέγεται **CleanData.py** . Το συγκεκριμένο, έχει ως λειτουργία να διαβάζει το αρχείο για την κάθε μετοχή να τα μετατρέπει σε DataFrame και να υπολογίζουν ένα μικρό growth approximation της μετοχής από την 1η της μέρα ως την τελευταία της.

Το σκορ αυτό υπολογίζεται με το **Close τελευταίας μέρας/ Close 1ης μέρας** . Τα σκορ αυτά σώνωνται σε ένα dictionary με key = filename και value = score .

Στη συνέχεια, διαλέγω τα N αρχεία με το καλύτερο score. Τα συγκεκριμένα αρχεία τα αντιγράφω σε ένα φάκελο **cleanstocks.** Με αυτον τον τρόπο διαλέγουμε N μετοχές ανάλογα με το πόσες μετοχές θέλουμε να φορτώσουμε.

```
import pandas as pd
import os
from collections import OrderedDict
from operator import itemgetter
from itertools import islice
import shutil

os.chdir('stocks')
filenames = [x for x in os.listdir() if x.endswith('.txt') and os.path.getsize(x) > 0] # al

myScoreDict = {}

for filename in filenames:
    df = pd.read_csv(filename,sep=',')
    myScore = df.iloc[-1]['Close'] / df.iloc[0]['Close']

    myScoreDict[filename] = myScore

### εχω τα score στο dictionary. θα κάνω sort και θα παρω τα 200 πρώτα στο φάκελο CleanStoc
n = 5000 # αναλογα ποσα θελω να διαλεξω
myScoreDict = OrderedDict(sorted(myScoreDict.items(), key = itemgetter(1), reverse = True))
myStocks = list(islice(myScoreDict,n))

files = os.listdir()

for file in files:
    if file in myStocks:
        full_file_name = os.path.join('', file)
        if os.path.isfile(full_file_name):
            shutil.copy(full_file_name, '../cleanstocks')
```

concatenate.py

Στη συνέχεια, αυτά τα αρχεία που διαλέξαμε θέλουμε να τα ενώσουμε σε ένα DataFrame που θα είναι ταξινομημένο βάση ημερομηνίας.

Αυτή τη λειτουργία, την εκτελέι το αρχέιο **concatenate.py** . Αρχικά, διαβάζουμε το κάθε αρχείο από τον φάκελο **cleanstocks** και το αποθηκεύουμε σε ένα DF. Κάνουμε drop τη στήλη OpenInt και προσθέτουμε τη στήλη Stock με το όνομα της μετοχής σε κάθε γραμμή.

Η επόμενη πράξη, που κάνουμε πάνω στο DataFrame της μετοχής είναι να διαιρέσουμε τη στήλη Volume με το 10, ώστε να μη ξεπεράσουμε το όριο αγοραπωλησιών που δίνεται στην άσκηση. Θα δείτε παρακάτω στην εξήγηση της μεθοδολογίας για την εύρεση των ακολουθιών ότι κάνουμε μια κίνηση ανά μέρα, οπότε η συγκεκριμένη πράξη μας καλύπτει ως προς αυτή τη συνθήκη.

Μετά από όλες αυτές τις πράξεις προσθέτουμε το DataFrame της μετοχής σε μια λίστα , που θα περιέχει όλα τα Dataframes των μετοχών. Έπειτα, συνενώνω όλα τα Dataframes της λίστας με την εντολή concatenate και το καινούριο Dataframe που είναι η ένωση των άλλων το κάνουμε sort by date.

Τέλος, αποθηκεύουμε το συγκεκριμένο Dataframe ως **mystock.txt** το οποίο θα το διαβάζει το κυρίως μας πρόγραμμα **main.py**

```
import pandas as pd
import os
os.chdir('cleanstocks')


def stockmerge():

    filenames = [x for x in os.listdir() if x.endswith('.txt') and os.path.getsize(x) > 0]
    list_with_df = []
    for file in filenames:
        df = pd.read_csv(file, sep=',')
        df = df.drop(labels='OpenInt', axis=1)
        df['Stock'] = file.split('.')[0]
        df['Volume'] = df['Volume'] * 0.1
        df = df.astype({'Volume': 'int64'})
        list_with_df.append(df)


    result = pd.concat(list_with_df)
    result = result.sort_values(by =['Date'])
    os.chdir('..')
    result.to_csv('mystock.txt',sep=',')

    
stockmerge()
```

ΜΕΘΟΔΟΛΟΓΙΑ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΤΩΝ 2 ΑΚΟΛΟΥΘΙΩΝ

main.py και helper.py

Τα δύο αυτά αρχεία συνδυάζονται για να δώσουν την επίλυση στο πρόβλημα μας. Το main.py περιέχει την λογική του προγράμματος, ενώ το **helper.py** περιέχει συναρτήσεις που καλούνται από την **main.py** για συγκεκριμένες λειτουργίες , που θα αναλυθούν παρακάτω.

Αρχικά, στο **main.py** διαβάζουμε σε 1 DataFrame το αρχείο **mystock.txt.** Το αρχείο αυτό έχει τις εξής στήλες:

- df = |Date | Open | High | Low | Close | Volume | Stock |

Επιπλέον έχουμε και κάποια άλλα DataFrames τα οποία έχουν την εξής ονομασία και δομή.

1. Profit = |date | profit |
2. bal = |date | balance |
3. moves_history = | date | move | stock | Xvolume |
4. stocks_bought = | stocks | volume | pricepaid |

Τα παραπάνω DataFrames έχουν με τη σειρά τις ακόλουθες ιδιότητες.

1. Το **profit** αποθηκεύει μετά απο κάθε μας κίνηση την ημερομηνία που την κάναμε και το τρέχον διαθέσιμο πόσο που έχουμε για να αγοράσουμε μια άλλη μετοχή.
2. Το **bal** αποθηκεύει και αυτό μετά απο κάθε μας κίνηση την ημερομηνία που την κάναμε και την τρέχον αποτίμηση η οποία είναι profit(κέρδος) + αξία μετοχών που έχουμε * Volume που έχουμε στο close της μέρας που κάναμε την τελευταία μας κίνηση.
3. Το **moves_history** είναι αυτό που μας βοηθάει να γράψουμε τη δομή του αρχείου small.txt larget.txt, καθώς αποθηκεύουμε εκεί πέρα την ημερομηνία , το είδος της κίνησης , τη μετοχή που αγοράσαμε , και τον όγκο της μετοχής που αγοράσαμε.
4. Το **stocks_bought** dataframe περιέχει τις μετοχές που μου ανήκουνε , τον όγκο που κατέχω σε κάθε μετοχή και πόσα χρήματα έδωσα στην τελευταία μου αγορά της μετοχής εκείνης. Το τελευταίο χρησιμοποιείται κυρίως για να ξέρω να κάνω κινήσεις που θα μου προσφέρουν μεγαλύτερο κέρδος απο αυτό που έδωσα όταν τις αγόραζα.

Άλλες σημαντικές παράμετροι, για το τρέξιμο του αλγορίθμου είναι:

N_max → Μέγιστος αριθμός κινήσεων(1000 ή 1 000 000) ανάλογα την ακολουθία που θέλουμε.

T → Πόσες μέρες μπροστά να κοιτάμε απο την τελευταία μας κίνηση για να αποφασίσουμε την επόμενη

datetrack → Αποθηκεύω την ημερομηνία της τελευταίας κίνησης

daterange → Παράγει μια λίστα με ημερομηνίες από το datetrack εώς και T μέρες μπροστά.

Έμεις χρησιμοποιούμε το τελευτάιο στοιχείο της λίστας αυτής σαν όριο για το μέχρι ποια ημερομηνία να κοιτάμε για κάποια κίνηση.

Lastdate → επιβλέπει το daterange να μη ξεπεράσει την τελευτάια ημερομηνία του Dataframe και ξεφύγουμε

N → τρέχον αριθμός κινήσεων

```
import pandas as pd
from helper import  buy_low , sell_high , decide , write_to_file, update_balance,
pd.options.mode.chained_assignment = None


df = pd.read_csv('mystock2.txt',sep=',',index_col=0)
df = df.reset_index()
df = df.drop(labels='index', axis=1)


T = 14
N_max = 100000

datetrack = df.iloc[0]['Date']
lastdate = df.iloc[-1]['Date']
daterange =  pd.date_range(start=datetrack, periods=T) #period= 365 trexei
#if datetrack in daterange:
#    print('yo')

profit = {'date' :[datetrack],'profit':[1.0]}      # 1$ κερδος
profit = pd.DataFrame.from_dict(profit)

bal  = {'date' :[datetrack],'balance':[1.0]}     # αποτιμηση = κερδος + αξια μετο
bal = pd.DataFrame.from_dict(bal)

moves_history  = {'date': [], 'move': [], 'stock':[],'Xvolume':[]}
moves_history  = pd.DataFrame.from_dict(moves_history)

stocks_bought = {'stocks':[] , 'volume':[],'pricepaid':[]} # key is the stock val
stocks_bought  = pd.DataFrame.from_dict(stocks_bought)
```

Κύρια λογική του προγράμματος main.py .

Η λογική του προγράμματος είναι μέσα στο loop του main.py το οποίο τρέχει όσο δεν έχουμε ξεπεράσει τον αριθμό κινήσεων N_max και δεν έχουμε ξεφύγει εκτός απο την τελευταία ημερομηνία του που έχει το DataFrame df που περίεχει όλες τις μετοχές μας.

Καλούμε τη συνάρτηση decide, η οποία αποφασίζει για μια συγκεκριμένη χρονική περίοδο βασιζόμενη στις τίμες του datetrack και daterange του οποίου του 2ου η τιμή εξαρτάται όπως είπαμε από το T και το datetrack.

Αν η συνάρτηση αυτή γυρίσει κενό dataframe σημαίνει πως δεν έχουμε άλλες επιλογές κερδοφόρες οπότε σταματάει ο αλγόριθμος. Σε μια τέτοια περίπτωση, θα μπορούσαμε να το βάλουμε να αγοράζει απλά μετοχές που να μην μας προσφέρουν παραπάνω κέρδος και να τις πουλήσουμε αργότερα. Επομένως, η συνάρτηση επιστρέφει μόνο κερδοφόρες αποφάσεις.

Τρόπος λειτουργίας της decide.

Η συνάρτηση αυτή καλέι 2 άλλες συναρτήσεις που ονομάζονται to_buy , to_sell και αυτές επιστρέφουν ένα dataframe η κάθε μια για το ποιες μετοχές μπορείς να αγοράσεις ή να πουλήσεις αντίστοιχα σε μια συγκεκριμένη χρονική περίοδο ορισμένη από το datetrack και daterange.

Παρακάτω, βάζουμε ένα screenshot των 2 αυτών συναρτήσεων και μετά θα εμβαθύνουμε στη λειτουργία της decide.

```
#possible buy stocks
def to_buy(df,profit,datetrack,daterange):
    mask = (df['Date'] > datetrack)
    df_buy =  df[mask]
    df_buy = df_buy[df_buy['Date'] <= str(daterange[-1]).split()[0]]
    df_buy = df_buy[df_buy['Low'] <= profit['profit'].iloc[-1]]
    return df_buy

#possible sell stocks
def to_sell(df,stocks_bought,datetrack , daterange):
    if not(stocks_bought.empty):
        mask = (df['Date'] > datetrack)
        df_sell = df[mask]
        df_sell = df_sell[df_sell['Date'] <= str(daterange[-1]).split()[0]]
        df_sell = df_sell[df_sell['Stock'].isin(list(stocks_bought['stocks'].values))]
        return df_sell

    else:
        return pd.DataFrame()
```

Οπότε, τώρα έχοντας τα 2 dataframe, που μας δείχνουν τι μπορούμε να αγοράσουμε και τι μπορούμε να πουλήσουμε.

Αν το dataframe της συνάρτησης to_sell είναι άδειο τότε σίγουρα η decide θα ξέρει ότι πρέπει να αγοράσει. Στην περίπτωση αυτή γυρνάει ένα string “buy” και το dataframe από τη συνάρτηση to_buy.

Αλλιώς εάν, έχω να πουλήσω μετοχές , οπότε το dataframe που μας επιστρέφει η to_sell δεν είναι κενό, προσπαθώ να βρώ ποια απο αυτές τις μετοχές θα μου προσφέρει παραπάνω κέρδος από αυτό που πλήρωσα όταν την αγόρασα την τελευταία φορά.

Αν καμιά μετοχή δεν μπορεί να μου προσφέρει ένα κέρδος μεγαλύτερο από το ποσό που έδωσα όταν αγόραζα τη μετοχή τότε η συνάρτηση επιστρέφει πάλι ενα string “buy” και το dataframe με τις μετοχές που μπορώ να αγοράσω.

Αυτή είναι η λειτουργία της decide και τώρα θα δούμε λίγο αναλυτικότερα το πως αποφασίζει ποιες μετοχές έχουν να τις προσφέρουν παραπάνω κέρδος από το ποσό για να αποκτηθούν.

Ξεκινάμε με μια λούπα, για κάθε μετοχή που έχουμε στην κατοχή μας στο dataframe stocks_bought. Έπειτα, παίρνω κάθε φορά τα rows που ενδιαφέρουν τη συγκεκριμένη μετοχή από το dataframe που μας επέστρεψε η συνάρτηση to_sell. Επίσης, βρίσκουμε από το dataframe stocksbought πόσο όγκο έχουμε για τη συγκεκριμένη μετοχή και πόσα δώσαμε την τελευτάια φορά για να αγοράσουμε κάποιο μέρος αυτής της μετοχής(pricepaid).

Στη συνέχεια, έχοντας δημιουργήσει ένα μικρό dataframe που περίεχει έγγραφες μιας μετοχής απο αυτές που μπορούμε να πουλήσουμε, δημιουργώ μια στήλη τη VolumeToSell που έχει ως τιμή των όγκο που μπορώ να πουλήσω. Ουσιαστικά, αυτή η στήλη ελέγχει να μη πουλήσω παραπάνω από τον όγκο που μας επιτρέπει η μετοχή εκείνη τη μέρα.

Τέλος, δημιουργούμε 2 ακόμα νέες στήλες τις difference και earning.

Η difference είναι η διαφορά μεταξύ του πόσα θα βγάλω αν πουλήσω αυτήν την μετοχή με το volume που έχω με την τη μεταβλητή pricepaid.

Η earning είναι το κέρδος που μας αποφέρει αν πουλήσουμε τη μετοχή εκείνη τη συγκεκριμένη μέρα.

Από αυτά τα μικρά dataframe κρατάμε μόνο τις λίστες με difference θετικό.

Τέλος, αυτά τα μικρά dataframe τα βάζουμε σε μια λίστα , την οποία κάνουμε concatenate. Το νέο αυτό result dataframe είναι ουσιαστικά σαν αυτό που επέστρεψε η συνάρτηση to_sell ώστοσο με 3 επιπλέον στήλες τις volume2sell, difference , earnings , οι οποίες αργότερα θα μας βοηθήσουν στο να αποφασίσουμε ποια μετοχή θα πουλήσουμε και σε ποια ημέρα.

```python
#decide to sell
def decide(df,profit,stocks_bought,datetrack,daterange):
    df_buy = to_buy(df,profit,datetrack,daterange)
    df_sell = to_sell(df,stocks_bought,datetrack,daterange)
    if df_sell.empty:
        return "buy" , df_buy

    temp_dfs = []
    #df_sell['difference']= np.nan
    #df_sell['earning']= np.nan
    for stock in list(df_sell.Stock.unique()):
        temp = df_sell.loc[df_sell['Stock'] == stock]
        volume_bought = stocks_bought.loc[stocks_bought['stocks'] == stock]['volume'].values[0]
        pricepaid = stocks_bought.loc[stocks_bought['stocks'] == stock]['pricepaid'].values[0]
        temp.loc[temp['Volume'] >= volume_bought  , 'VolumeToSell' ] = volume_bought
        temp.loc[temp['Volume'] < volume_bought   , 'VolumeToSell' ] = temp['Volume']

        temp['difference'] =  (temp['High'] * temp['VolumeToSell'] )- pricepaid
        temp['earning'] =  temp['High'] * temp['VolumeToSell']
        temp = temp.loc[temp['difference'] > 0]
        temp_dfs.append(temp)
    result = pd.concat(temp_dfs)
    result = result.sort_values(by =['Date'])
    if result.empty:
        return 'buy' , df_buy
    else:
        return "sell", result
```

Περίπτωση Buy:

Καλώ τη συνάρτηση buy_low η οποία δέχεται σα παράμετρο το dataframe της decide που ουσιαστικά είναι αυτό που γυρνάει η συνάρτηση to_buy αν έχει αποφασιστεί ότι καλύτερη κίνηση είναι να αγοράσω μια μετοχή.

Αυτή η συνάρτηση αποφασίζει ποια μετοχή θα αγοράσω ως εξής:

Δημιουργούμε μια στήλη variance η οποία είναι η διαίρεση των στήλων High/Low και κάνουμε χρήση της groupby βάση της στήλης stock και ζητάμε το μέσο όρο των variance για κάθε μετοχή. Ουσιαστικά , έτσι βρίσκουμε τη μετοχή που έχει τον καλύτερο συνδυασμό High/Low.

Επομένως , έχοντας βρει ποια μετοχή μας ενδιαφέρει , αγοράζουμε τη συγκεκριμένη μετοχή τη μέρα που το Low της είναι πιο χαμηλά από τα άλλα.

```
def buy_low(dcion_df):
    # get the row of min value in column Low
    dcion_df['variance'] = dcion_df['High'] / dcion_df['Low']
    groupby = dcion_df.groupby(['Stock'])['variance'].mean()
    groupby = groupby.sort_values(ascending=False)
    a = groupby.idxmax() # stock i want!
    stock_interested = dcion_df[dcion_df['Stock'] == a]

    return stock_interested.loc[stock_interested['Low'].idxmin()]
```

Τέλος αφού έχουμε αποφασίσει τι θα αγοράσουμε κάνουμε update όπως και στην προηγούμενη περίπτωση τις μεταβλητές datetrack, profit , move_history και stocks_bought ανάλογα με τα δεδομένα μας.

Γενικά, από μια μετοχή που επιλέγω να αγοράζω εξαντλητικά, δηλαδή όσο αντέχει το ποσό που έχω, αλλιώς εάν μπορώ να αγοράσω παραπάνω από όσο όγκο έχει , τότε απλά αγοράζω όλη τη μετοχή.

```
if decision == "buy": #decision_df comes from to_buy function.
    best_choice = buy_low(decision_df) # best choice to buy
    #now i need to find how much i will buy from that stock!
    #         update datetrack , profit , moves  and , stocks_bought
    datetrack = best_choice['Date']
    if best_choice['Low'] <= 0:
        xvol = int(best_choice['Volume'])
    else:
        xvol = int (profit['profit'].iloc[-1] // best_choice['Low'] )
        if xvol > best_choice['Volume']:
            xvol = int(best_choice['Volume'])
    pricepaid = xvol * best_choice['Low']
    new_profit =  profit['profit'].iloc[-1] - pricepaid
    profit = profit.append({'date' : best_choice['Date'], 'profit':new_profit } ,ignore_index=True)

    moves_history = moves_history.append({'date' : best_choice['Date'], 'move':'buy-low','stock':best_choice['Stock'],'Xvolume':xv

    if best_choice['Stock'] in stocks_bought.stocks.unique():
        i = stocks_bought.index[stocks_bought['stocks'] == best_choice['Stock'] ].tolist()[0]
        stocks_bought.at[i,'volume'] += xvol
        stocks_bought.at[i,'pricepaid'] = pricepaid
    else:
        stocks_bought = stocks_bought.append({'stocks' : best_choice['Stock'], 'volume':xvol,'pricepaid':pricepaid } ,ignore_index
```

Περίπτωση Sell:

Καλώ τη συνάρτηση sell_high η οποία έχει ως input το dataframe που γυρνάει η decide στην περίπτωση του sell , το οποίο είναι οι μετοχές που μπορώ να πουλήσω και μου προσφέρουν παραπάνω κέρδος από το ποσό που έδωσα όταν τις αγόρασα την τελευταία φορά.

```
def sell_high(dcion_df):
    # get the row of max value in column earning
    return dcion_df.loc[dcion_df['difference'].idxmax()]
```

Η συνάρτηση αυτή επιστρέφει την πιο κερδοφόρα μετοχή για να πουλήσουμε. Στη συνέχεια κάνω update τη μεταβλητή datetrack βάση της ημερομηνίας που έκανα την πώληση, ανανεώνω το profit μου , αποθηκεύω την κίνηση στο moves_history dataframe και τέλος κάνω update το stokcs_bought dataframe ανάλογα με τα νέα μου δεδομένα.

```
if decision == "sell":
    best_choice = sell_high(decision_df)
    #change profit , stocks_bought , moves_history, datetrack
    datetrack = best_choice['Date']

    new_profit = profit['profit'].iloc[-1] + best_choice['earning']
    profit = profit.append({'date' : best_choice['Date'], 'profit':new_profit } ,ignore_index=True)

    xvol =  int(best_choice['VolumeToSell'])


    moves_history = moves_history.append({'date' : best_choice['Date'], 'move':'sell-high','stock':best_choice['Stock'],'Xvolume':

    i = stocks_bought.index[stocks_bought['stocks'] == best_choice['Stock'] ].tolist()[0]
    stocks_bought.at[i,'volume'] -= xvol

    stocks_bought = stocks_bought[stocks_bought.volume != 0] # remove row with that stock cause i sold it.
```

Τέλος κίνησης:

Στο τέλος μετά απο κάθε κίνηση , αυξάνω τον αριθμό κινήσεων κατά 1 και τυπώνουμε τον αριθμο κινήσεων και το χρηματικό ποσό που έχουμε διαθέσιμο(profit).

Επίσης, ενημερώνουμε το datetrack και το daterange.

Πολύ σημαντικό είναι μετά από κάθε κίνηση να γίνει ανανέωση του dataframe για το balance →αποτίμηση.

Το balance υπολογίζεται στην παρακάτω συνάρτηση:

```
def update_balance(df,stocks_bought,datetrack, profit):
    balance_sum = profit
    mystocks = list(stocks_bought.stocks.unique()) # oi metoxes pou exw
    myvolume = list(stocks_bought.volume.unique()) # to volume pou exw gia kathe mia apo auti
    myStockVol = zip(mystocks,myvolume)            # tuples (stockName,Volume)

    for i,vol in myStockVol:
        #i want the row with stock name = i , and date = datetrack
        myrow = df.loc[(df['Date'] == datetrack) & (df['Stock']== i)]
        if not(myrow.empty):
            close_price = myrow['Close'].values[0]
            balance_sum += vol * close_price
        else:
            continue


    return  balance_sum
```

Παρατηρούμε ότι αρχικοποιούμε το balance_sum με το profit. Έπειτα, δημιουργώ μια λίστα με tuple που έχει τη μετοχή που μου ανήκει και τον αντίστοιχο όγκο

Για κάθε μια απο αυτές τις μετοχές ψάχνω στην ημερομηνία που έκανα την τελευταία μου κίνηση πόσο ήταν το Close της και υπολογίζω το Close* τον όγκο που έχω και το προσθέτω κάθε φορά στη μεταβλητή Balance_Sum.

Αν μια μέρα δεν υπήρχε η εγγραφή για μια μετοχή τότε απλώς το προσπερνάω και το αφήνω ως έχει.

Τέλος αλγορίθμου ~ Σχεδίαση διαγραμμάτων και αρχείου txt με κινήσεις.

Η δημιουργία του txt γίνεται με την ακόλουθη συνάρτηση από το helper.py

```
def write_to_file(N,moves_history):
    if N<=1000:
        f = open("small.txt", "w")
    else:
        f = open("large.txt", "w")
    f.write(str(N))
    f.write("\n")
    for i in range(len(moves_history)):
        f.write(moves_history.loc[[i]]['date'].values[0])
        f.write(" ")
        f.write(moves_history.loc[[i]]['move'].values[0])
        f.write(" ")
        f.write(moves_history.loc[[i]]['stock'].values[0])
        f.write(" ")
        f.write(str(int(moves_history.loc[[i]]['Xvolume'].values[0])))
        f.write("\n")
    f.close()
```

Η συνάρτηση αυτή δέχεται των αριθμό των κινήσεων που κάναμε και το dataframe move_history

Παρακάτω δείχνουμε τη συνάρτηση για τη σχεδίαση διαγράμματος αποτίμησης-κέρδους.

```
#plot the diagram of profit and bal through time!
def plot_profit_balance(profit,bal):
    del bal['date']
    df_merged = pd.concat([profit, bal], axis=1)
    # clean date column from y-m-d to y. !
    df_merged['date'] = pd.DatetimeIndex(df_merged['date']).year
    ax = df_merged.set_index('date').plot(figsize=(10,5), grid=True, kind= 'area',legend=True , stacked=True, title = 'Di
    ax.margins(0,0)
    plt.show()
```

Η συνάρτηση αυτή δέχεται σαν όρισμα το profit , bal dataframes ενώνω αυτά τα 2 dataframe ώς προς τις στήλες και διαγράφω την μια date η οποία μετά την ένωση υπάρχει δύο φορές.

Στη συνέχεια κάνω το διάγραμμα με άξονα χ το Year από κάθε date και άξονα Υ τις 2 άλλες στήλες. Το διάγραμμα είναι τύπου area.

ΑΠΟΤΕΛΕΣΜΑΤΑ: ΑΚΟΛΟΥΘΙΕΣ & ΔΙΑΓΡΑΜΜΑΤΑ

Βρίσκονται στο φάκελο **sequence_graphs**.

Για το small.txt μπορούμε να θέσουμε T = 1500 στο main.py ή κάτι αντίστοιχο και θα βγει η εξής ακολουθία.

23

1962-06-25 buy-low ge 2

1965-09-27 sell-high ge 2

1966-09-08 buy-low ge 3

1967-09-18 sell-high ge 3

1970-07-01 buy-low ba 9

1972-12-12 sell-high ba 9

1974-12-06 buy-low c 1

1978-08-10 sell-high c 1

1982-08-12 buy-low apa 8

1983-08-22 sell-high apa 8

1986-11-28 buy-low rost 337

1989-10-10 sell-high rost 337

1993-04-12 buy-low iac 365

1995-11-28 sell-high iac 365

1998-10-08 buy-low ebay 7706

2000-03-27 sell-high ebay 7706

2002-08-20 buy-low axgn 1848

2005-02-14 sell-high axgn 1848

2006-02-22 buy-low wpm 107867

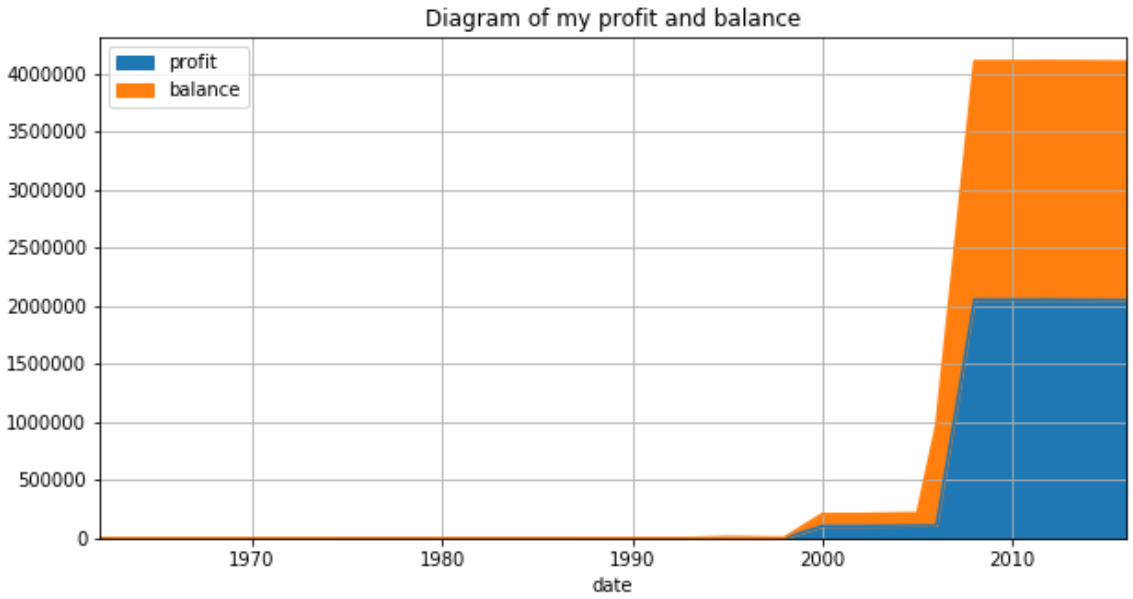
2008-03-14 sell-high wpm 107867

2010-08-24 buy-low flat 10

2012-12-06 sell-high flat 10

2016-06-29 buy-low smbk 330

Με το εξής διάγραμμα αποτίμησης-κέρδους.



Τσεκάρουμε το αρχείο στο validator:

Success:

After 23 transaction(s), your profit is 2047626.33461 dollars.

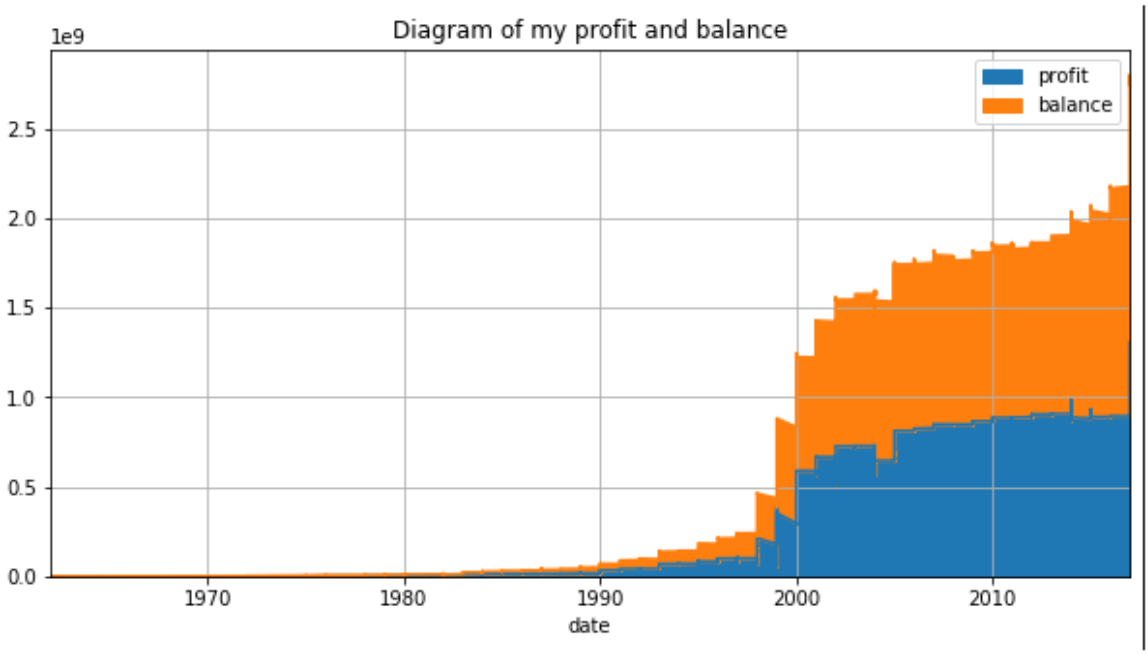
Για τη μεγάλη ακολουθία κινήσεων, λόγω έλλειψης υπολογιστικής δύναμης, επειδή στις 1000 κινήσεις και μετά διαρκεί αρκετές ώρες παρουσιάζω μια λύση για T=14 , δηλαδή να βλέπει 14 μέρες.

Success:

After 3184 transaction(s), your profit is 1304320825.6931992 dollars.

Η συγκεκριμένη ακολουθία είναι στο φάκελο sequence_graphs με όνομα large.txt

Παρακάτω φαίνεται το διάγραμμα της.



Για να υλοποιηθούν κινήσεις επιπέδου 10 000 ή 20 000 το T θα πρέπει να αποκτά τιμές ανάμεσα στο 1-3.

Γιατί για παράδειγμα αν το βάλουμε να αποφασίζει ανά μέρα τότε εφόσον εχουμε 55 χρόνια τότε ο μέγιστος αριθμός κινήσεων που θα κάνει θα είναι 55*365= 20075.

Απλά λόγω έλλειψης υπολογιστικής δύναμης παρουσιάζω ένα μικρότερο κομμάτι.